REMARKS

In the above-identified Office Action, the Examiner objected to the DRAWINGS and to the SPECIFICATION. Claims 1 - 23 were rejected under 35 U.S.C. §112, second paragraph, for indefiniteness. Claims 16 - 19 were rejected under 35 U.S.C. §101 because they are directed to non-statutory subject matter. Claims 1 – 8, and 10 were rejected under 35 U.S.C. §102(b) as being anticipated by Fogel et al. Claims 11 – 15 and 20 – 23 were rejected under 35 U.S.C. §102(b) as being anticipated by Bloom. Claims 9 and 16 – 19 were rejected under 35 U.S.C. §103(a) as being unpatentable over Fogel et al. in view of Cvsln.

In reviewing the SPECIFICATION, Applicants have encountered a few typographical/grammatical errors, including the "if $a^1 0$" typographical error mentioned by the Examiner. These errors have been corrected.

In response to the objection to the DRAWINGS, a REPLACEMENT Fig. 6 is filed concurrently with this Response. In this REPLACEMENT Fig. 6, step 620 is expanded into new steps 612, 686 and 688 in order to show the condition to go from step 620 to step 690 required by the Examiner. Likewise, step 640 is expanded into steps 630, 682 and 684 so that the condition to go from step 640 to step 620 is detailed out. The SPECIFICATION has accordingly been updated to explain the added new steps in the REPLACEMENT Fig. 6.

By this amendment to the SPECIFICATION and DRAWINGS, no new matter has been added to the Application since the added steps are implied in step 620 (i.e., for each source unit) and step 640 (for each construct). Applicants believe that the objection to the DRAWINGS and SPECIFICATION has been overcome and kindly request its withdrawal.

Although contrary to the Examiner's assertion, the SPECIFICATION does provide a standard for ascertaining the requisite degree of the word "similar" (see pages 14 – 16), Claims 1 – 17 and 20 – 23 have been nonetheless amended, where appropriate, to replace the word "similar" with the word "derived".

Likewise, Claims 3 – 7 and 10 – 15 have been amended to replace, where appropriate, the term "reasonable size" with the term "at least a threshold size" or "at least a minimal threshold size". This replacement term can be found throughout the SPECIFICATION and particularly in the paragraph that starts with "Figure 4 further elucidates" on page 11.

The terms "N tokens", "M tokens" and "P tokens" in claims 11 – 17 and 20 – 23 have been defined or deleted. For example, the term "the first N tokens" in Claim 11 is changed to the term "N tokens" and N is defined as a positive integer. In that claim M is made to equal to N to define M. In Claims 20 – 23, the terms "N tokens" and "M tokens" are deleted so is the term "P tokens" in Claims 16 and 17.

Also, the terms "too small" and "too large" in claims 11 – 15 are deleted from the claims (see Claim 11). The term "owners" or "owner" in Claims 1, 10, 14, 15, 18 and 22 is replaced by the term "a user responsible for" or "users responsible for", respectively. These replacement terms can be found in the last sentence of the TECHNICAL FIELD section on page 1 as well as the first sentence on page 11. Note that "user" is used instead of "developer" as per its use in the third sentence in the first paragraph of the BACKGROUND OF THE INVENTION section on page 1.

All claims (i.e., Claims 1, 2, 10 and 22) that have been rejected under 35 USC §112, second paragraph, because of the use of a term that lacks proper antecedent basis have been accordingly amended to overcome the rejection.

Thus, by the amendments to the claims, as explained above, all 35 USC §112, second paragraph, rejections have been cured. Hence, Applicants respectfully request withdrawal of the rejection.

Regarding the 101 rejection of Claims 16 – 19, Applicants have amended elements of the claims to specify that everything that is being done is being done under the control of a processor. Thus, Applicants believe that the 101 rejection has been overcome and thus request its withdrawal.

By this amendment, Claims 1 – 23 remain pending in the Application. For the reasons stated more fully below, Applicants submit that the pending claims are allowable over the applied references. Hence, reconsideration, allowance and passage to issue are respectfully requested.

As disclosed in the SPECIFICATION, with an object-oriented programming language, for example, a program is constructed from a number of "objects," each of which includes data and/or one or more sets of instructions that define specific operations to be performed on the data. A few or a large number of components may be used to create an object, and a large number of objects may be used to build a computer program with each object interacting with other objects in the computer program to perform desired operations. When one object invokes a particular routine in another object, the former object is often said to be calling the routine in the latter object. Some general purpose objects in a computer program may support basic operations, e.g., displaying information to a user, printing information on a printer, storing or retrieving information from a database, etc. Particularly, these generic types of objects are called by many different objects so that a change in the code in one of the objects may benefit other similar or related objects, either from which the edited object or component was derived, or from other objects or components which were derived from the edited object.

Despite the fact that components are intended to be reused, programmers often fail to take advantage of these packaging techniques and, instead, copy code from one component to create a similar one. Finding all of the components or objects which are derived in this way can be incredibly difficult, if not impossible. Examining hundreds or thousands of lines of program instructions is tedious and time consuming, and sometimes variable names change or strings change, so finding related or derivative sections of code is not easily accomplished. Manual review of the code, thus, is fraught with the possibilities that not all related or derived components or objects will be located.

A computer application, including objects, typically has hundreds or thousands of these components, which in turn may be grouped into smaller pieces of source code called program structures or constructs, as is known in the field. A conditional construct specifies several different execution sequences, for example, a CASE statement, an IF statement, a conditional expression in ALGOL. An executable construct specifies one or more actions to be taken by a computer program at execution time and comprise executable statements. A loop construct specifies an iteration in the execution sequence, for example, DO loops in FORTRAN, FOR loops in ALGOL, PERFORM loops in COBOL, DO WHILE loops in PL/I. There are other constructs that exist and still others continually arising as new languages arise, especially as Internet-based applications become plentiful.

There is a need in the industry to help programmers locate related or derived components and objects when editing a particular component or code. There is a further need in the industry to determine if the changes made to one component or object should be applied to the other related or derived components or objects. The present invention satisfies this need.

According to the teachings of the invention, a source code that has been edited is recognized, then a program construct in the edited source code is identified. After identifying the program construct, a construct list of other constructs having similar and/or related code to that which was edited, is determined. Then, the program construct in the edited source code is compared with the other constructs to determine a degree of similarity, and if the degree of similarity is equal to or beyond a threshold of similarity, then owners (i.e., users responsible to maintain the constructs or code) of the other constructs are notified.

In order to maintain a reasonable number of other constructs that might be enhanced by a change to the edited construct, the algorithm determines what is a reasonable size and then looks only to those constructs having a reasonable size for placement in the construct list. Tokens of both the edited construct and the other constructs determined to be of a reasonable size are parsed. The parsed tokens of the edited source code are compared with the parsed tokens of the other constructs in the construct list. The parsed tokens may be weighted during comparison so that a degree of similarity can be established, the weights based on type, name, and/or representation. The sum of the weights of the compared tokens is evaluated against a threshold of similarity. The construct list is stored for future use.

The invention is set forth in claims of varying scopes of which Claims 1 and 10 are illustrative.

> 1. An algorithm to improve efficiency of editing source code, comprising
> recognizing that a source code has been edited;
> ***identifying a program construct having the edited source code***;
> ***constructing a construct list of at least one other construct having derived and/or related code to the program construct***;
> ***comparing the at least one other construct with the program construct having the edited source code***; and
> ***if, in response to comparing the at least one other construct with the program construct, a commonality between the at least one other construct and the program construct is found to be equal to or beyond a threshold of similarity, then notifying a user responsible for the at least one other construct that the source code of the program construct has been edited***. (Emphasis added.)

> 10. An efficiency algorithm to improve efficiency of editing source code, comprising
> recognizing that a source code has been edited;
> ***identifying a program construct having the edited source code and parsing tokens of the edited source code***;
> ***constructing a construct list of at least one other construct of at least a minimal threshold size having related code by parsing a sequence of tokens***

> ***from each of a plurality of constructs of the at least minimal threshold size***;
>
> ***comparing the parsed tokens of the edited source code with the parsed tokens of each of the plurality of constructs in the construct list, and weighting the compared tokens***;
>
> ***summing the weights of the compared tokens to determine if the sum is equal to or beyond a threshold of similarity, and if so, then determining if a user responsible for the at least one other construct is to be notified***; and
>
> ***storing the construct list***. (Emphasis added.)

The Examiner rejected independent Claims 1 and 10 under 35 U.S.C. §102(b) as being anticipated by Fogel et al., Applicants respectfully disagree.

Fogel et al. disclose a manual that describes how to use and administer CVS (Concurrent Versions System), a free software, for collaboration and version control. In the manual, it is disclosed that CVS: (1) enables a plurality of developers to edit a program simultaneously, (2) assumes the burden of integrating all the changes in the program, and (3) keeps track of any conflicts by using the copy-modify-merge model. For example, developer A and developer B may each request a working copy of a program from CVS. Both developers may edit freely their working copy. When a first developer (e.g., developer A) is done, the first developer may commit the changes made to the program into CVS. CVS then incorporates the changes into the program. When the second developer (e.g., developer B) is done, the second developer may also commit the changes made to the program into CVS. However, before the changes are incorporated into the program, CVS will compare the changed copy of developer A with the changed copy of developer B. If there is a conflict (e.g., if both developers change the same lines of code), CVS will flag the change in the copy of developer B. At this point, it is up to developer B, with or without the help of developer A, to resolve the conflict. When the conflict is resolved, developer B may then retry committing the changes to the program by again submitting the edited copy of the program to CVS. If there continues to be a conflict, the working copy of developer B will be rejected and the conflicting code flagged as before. If, on the other hand, there is not a conflict, the changes made by developer B will now be incorporated into the program. Thus, the program will now iclude the changes from developers A and B.

Hence, CVS compares copies of files to determine whether (1) there are changes in the copies of the files and if at least two copies of the files have changes therein then (2) whether the changes in one copy conflict with the changes in the other copy. If so, then (3) the conflict will be flagged and (4) a user notified of the conflict.

By contrast, the claimed invention compares a changed construct in a file (that is already known to have had changes therein) with constructs in the same file and/or other files that have not been changed for instance. The comparison is done to determine whether there are constructs in the other files that were derived from the changed construct. If so, then a user responsible for the other files or constructs may be notified in order for the user to accordingly make changes in those constructs derived from the changed construct.

Note that a construct is a small piece of source code which makes up, for instance, a CASE statement or an IF statement or a DO loop in a program (see last paragraph on page 2).

Thus, Fogel et al. do not teach, show or so much as suggest the steps of (1) ***identifying a program construct having the edited source code***; (2) ***constructing a construct list of at least one other construct having derived and/or related code to the program construct***; (3) ***comparing the at least one other construct with the program construct having the edited source code***; and (4) ***if, in response to comparing the at least one other construct with the program construct, a commonality between the at least one other construct and the program construct is found to be equal to or beyond a threshold of similarity, then notifying a user responsible for the at least one***

other construct that the source code of the program construct has been edited as claimed in claim 1.

Further, Fogel et al. do not teach, show or suggest the steps of (1) ***identifying a program construct having the edited source code and parsing tokens of the edited source code***; (2) ***constructing a construct list of at least one other construct of at least a minimal threshold size having related code by parsing a sequence of tokens from each of a plurality of constructs of the at least minimal threshold size***; (3) ***comparing the parsed tokens of the edited source code with the parsed tokens of each of the plurality of constructs in the construct list, and weighting the compared tokens***; (4) ***summing the weights of the compared tokens to determine if the sum is equal to or beyond a threshold of similarity, and if so, then determining if a user responsible for the at least one other construct is to be notified***; and (5) ***storing the construct list*** claimed in Claim 10.

Independent Claims 11 and 20 were rejected under 35 U.S.C. §102(b) as being anticipated by Bloom. Again, Applicants respectfully disagree.

Bloom purports to teach a source code comparator computer program. According to the teachings of Bloom, the comparator computer program compares two versions of a source program and identifies the difference between the two. The program compares the two versions until a noncomparison (i.e., a difference) is detected. A search is then performed for a subsequent piece of code that is similar in the two versions. An alike sequence such as a symbolic address in both source programs is determined and used as a base from which another noncomparison or difference is determined by working backwards from the base. The smallest area of noncomparison (i.e., the smallest difference) in the two searches defines a difference between the source programs. The difference in a program is defined as an addition, deletion or modification by examining the statements within the change area. A search is made for a comparison. All statements preceding the comparison in the base reference file are marked as deletions. All statements preceding the comparison in the revised version are marked as additions. A comparison of a shortened portion of any statement is marked as a modification. After all comparisons in the change area are searched to define the changes, the program returns to the initial compare subroutine until the next noncomparison is detected and the process is repeated.

The Examiner stated that the first step in Fig. 1 (i.e., locate base module) is the equivalent of identifying a first construct. Applicants disagree.

A base module is defined by Bloom as a portion of the memory store that has the base reference source programs stored therein (see col. 3, lines 14 – 17). As mentioned above, a construct is a small piece of source code which makes up, for instance, a CASE statement or an IF statement or a DO loop (see last paragraph on page 2). Accordingly, the base module in the disclosure of Bloom is not a construct as defined in the SPECIFICATION.

Thus, Bloom does not teach show or suggest the steps of (1) ***identifying a first construct***; and (3) ***identifying a plurality of other constructs in the repository*** as claimed in Claim 11.

Further in the claimed invention, the comparison is being done between a first construct and a **plurality of other constructs**. By contrast the comparison in the disclosure of Bloom is between a first version of a program and a second version of the program not between a first version of the program and a plurality of other versions of the program. Thus, Bloom does not teach the steps of (4) ***parsing M tokens <u>of each one of the other constructs</u>, where M = N***; (5) ***comparing the M tokens <u>of each one of the other constructs</u> with the N tokens of the first construct***.

More over, nowhere in the disclosure of Bloom is there a reference to using weights in doing the comparison. Hence, Applicants submit that Bloom does not teach the steps of (6) ***determining a weight for each one of the N and M tokens based on name, type, and/or representation; (7) summing the weights of the N and M tokens***; (8) ***determining whether or not the sum of the weights of the M tokens meets or exceeds a threshold of similarity, the***

threshold of similarity being based on a percentage of the sum of the weights of the M tokens to the sum of the weights of the N tokens; and (9) ***identifying each construct whose sum of the weights of the M tokens meets or exceeds the threshold of similarity as being related to the first construct*** as claimed in independent Claim 11.

Independent Claim 20 includes the limitations of: (1) determining that a source code has been edited in an environment of computer program development; (2) determining if the edited source code is within a construct of size of a particular range; (3) parsing the construct having the edited source code if the edited source is within a construct of a size of a particular range; (4) finding and parsing other constructs in the environment having a size within the particular range; (5) creating a construct list of other constructs in the environment having a size within the particular range; (6) comparing tokens of the construct having the edited source code with tokens of the other constructs in the construct list; and (7) determining that the construct having the edited source code is related to any one of the constructs in the construct list if the tokens of any one of the constructs in the construct list equal the tokens of the construct having the edited source code.

As mentioned above, Bloom does not teach, show or suggest constructs as defined in the Application. Therefore, Bloom does not teach the steps of: (2) determining if the edited source code is within a **construct** of size of a particular range; (3) parsing the **construct** having the edited source code if the edited source is within a construct of a size of a particular range; (4) finding and parsing other **constructs** in the environment having a size within the particular range; (5) creating a **construct list of other constructs** in the environment having a size within the particular range; (6) comparing tokens of the **construct** having the edited source code with tokens of the **other constructs in the construct list**; and (7) determining that the **construct** having the edited source code is related to any one of the **constructs in the construct list** if the tokens of any one of the constructs in the construct list equal the tokens of the construct having the edited source code.

Independent Claims 16 and 18 were rejected under 35 U.S.C. §103(a) as being unpatentable over Fogel et al. in view of CvsIn. In support of the rejection the Examiner stated that Fogel et al. teach the claimed invention except that they (fogel et al.) do not explicitly disclose that the process is being performed in an integrated development environment (IDE). However, the Examiner continued, CvsIn does disclose integration of CVS in an IDE. Consequently, the Examiner reasoned, it would have been obvious to combine the teachings of Fogel et al. with those of CvsIn to arrive at the claimed invention. Applicants disagree.

Independent Claim 16 includes the limitations of: (1) a repository of source code into which programs in the integrated development environment are stored; (2) a constructor to determine, when executed by a processor, whether or not a construct in an edited program has been edited; (3) a construct list within the repository into which all constructs that are derived and/or related to the edited construct are listed; (4) a parser to parse the edited construct and the derived and/or related constructs, when executed by a processor; (5) a matchmaker to determine, when executed by a processor, all the constructs that are derived and/or related to the edited construct and to enter all constructs determined to be derived and/or related to the edited construct in the construct list; and (6) an announcer to notify, when executed by a processor, any of a plurality of programmers accessing the integrated development environment that the edited construct has been edited and the constructs that are derived and/or related to the edited construct.

As mentioned in the analysis of independent Claim 1 above, Fogel et al. do not teach, show or suggest the concept of a construct. Therefore, Fogel et al. do not teach (2) **a constructor to determine, when executed by a processor, whether or not a construct in an edited program has been edited**; (3) **a construct list within the repository into which all constructs that are derived and/or related to the edited construct are listed**; (4) **a parser to**

parse the edited construct and the derived and/or related constructs, when executed by a processor; (5) **a matchmaker to determine, when executed by a processor, all the constructs that are derived and/or related to the edited construct and to enter all constructs determined to be derived and/or related to the edited construct in the construct list**; and (6) **an announcer to notify, when executed by a processor, any of a plurality of programmers accessing the integrated development environment that the edited construct has been edited and the constructs that are derived and/or related to the edited construct**.

Thus, Applicants submit that independent Claim 16 is patentable over the combined teachings of Fogel et al. and CvsIn.

Independent Claim 18 also includes the limitations of a construct. Based on the analysis of Claim 16 immediately above, Applicants submit that Claim 18 is also patentable over the combined teachings of Fogel et al. and CvsIn.

Since all the independent Claims in the Application are patentable over the applied references, Applicants submit that their dependent claims are also allowable over the applied references. Consequently, Applicants once more respectfully request reconsideration, allowance and passage to issue of the claims in the application.

Respectfully Submitted

By: [illegible]
Volel Emile
Attorney for Applicants
Registration No. 39,969
(512) 306-7969